Master Thesis Proposals

June 2021

LAB VLSI

Politecnico di Torino

1859

People
VLSI LAB

NANOCOMPUTING

Gianluca Piccinini

Mariagrazia Graziano

Giovanni Amedeo Cirillo

Yuri Ardesi

Giuliana Beretta

Fabrizio Riente

Luca Gnoli.

Chiara Spano Elef.

Fabrizio Ro

Giovanna Turvani

Fabrizio Ottati

Marco Vacca

Andrea Coluccio

Mario Simoni.

Maurizio Zamboni

Andrea Marchesin

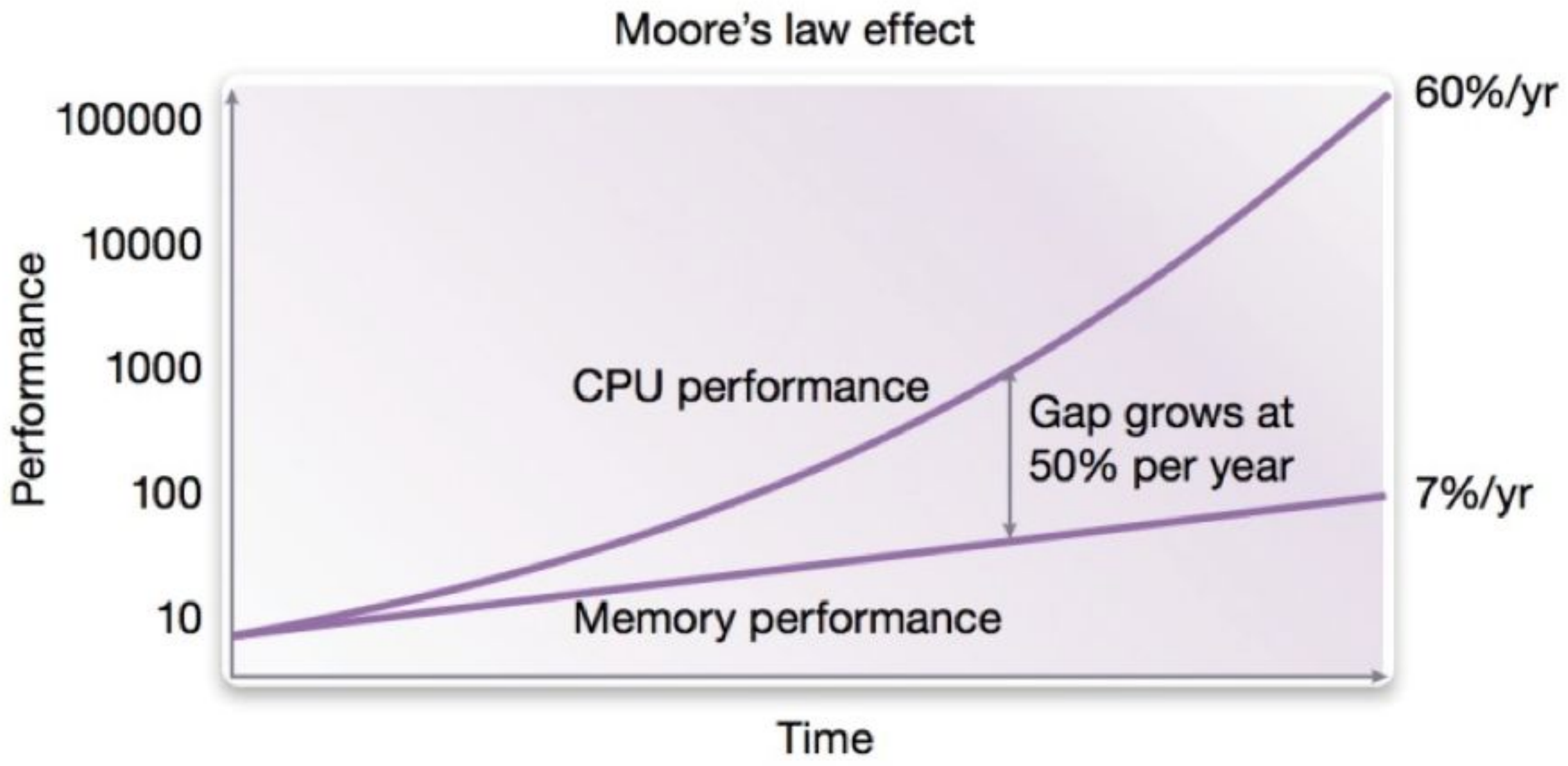
Umberto Garlando

Outline

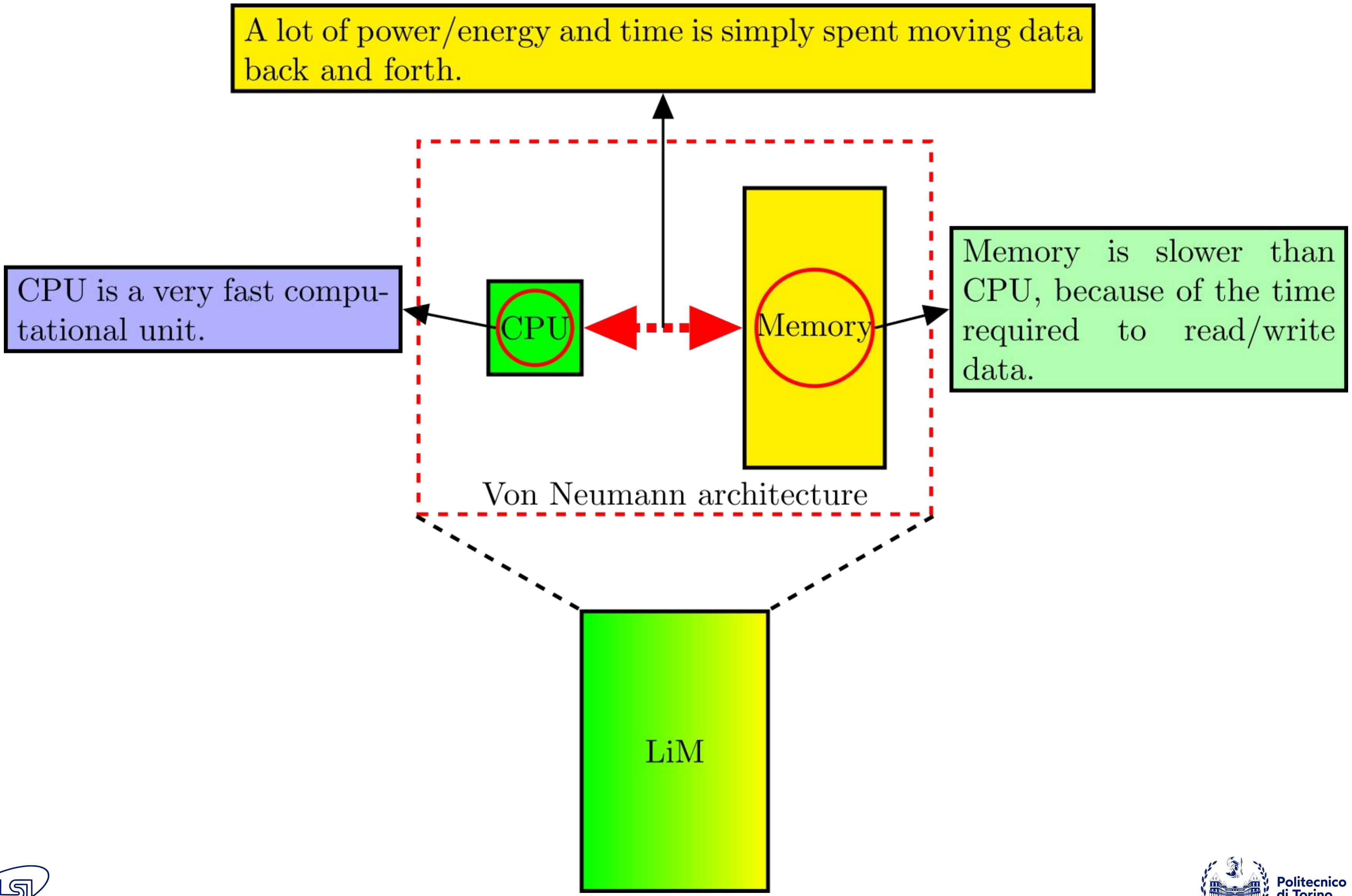
- **Logic In Memory**
  - T.LiM.1 : Architectural Explorations with Dexima CAD
  - T.LiM.2 : FPGA LiM Implementation
  - T.LiM.3 : LiM Toolchain
  - T.LiM.4 : Octantis
  - T.LiM.5 : Octantis & The LiM Toolchain
- **Quantum Computing**
  - T.QC.1 : Emulation of Quantum Annealing on GPU
  - T.QC.2 : Emulation of Quantum Gate Array On FPGAs
  - T.QC.3 : Emulation of Quantum Gate Array On GPUs
  - T.QC.4 : Compilation of Quantum Circuits
  - T.QC.5 : Quantum High Level Synthesis
- **CAD Tools**
  - T.CAD.1 Hybrid CMOS Simulation
  - T.CAD.2 ToPoliNano Simulator
- **Field Coupled Technologies**
  - TMAG.1 Racetrack memory architecture

Logic-In-Memory

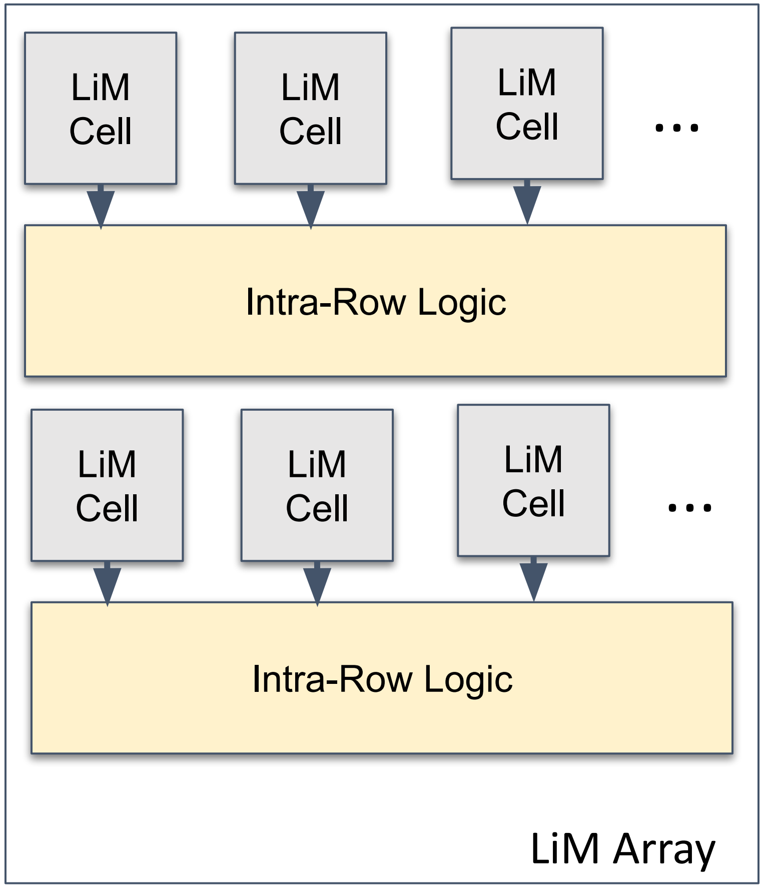
The memory bottleneck problem



What is Logic-in-Memory?


A lot of power/energy and time is simply spent moving data back and forth.
CPU is a very fast computational unit.
CPU
Memory
Memory is slower than CPU, because of the time required to read/write data.
Von Neumann architecture
LiM

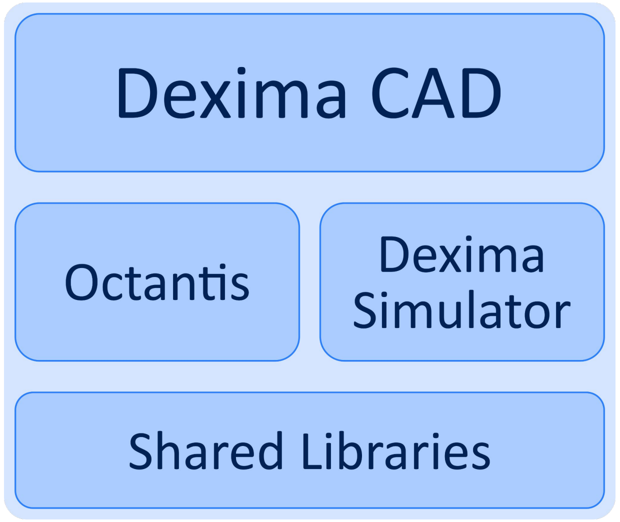
LiM Architectures



- **Parallel** computing capabilities to enhance performance
- In memory computing features, to minimize **power** consumption due to **data movement** and to increase **performance**
- **Modularity** to adapt processing capabilities depending on the application
- **Flexibility** to support for a wide range of data processing related operations

LiM Design Environment

- To design and test LiM architectures an **entire Software Toolchain** has been created

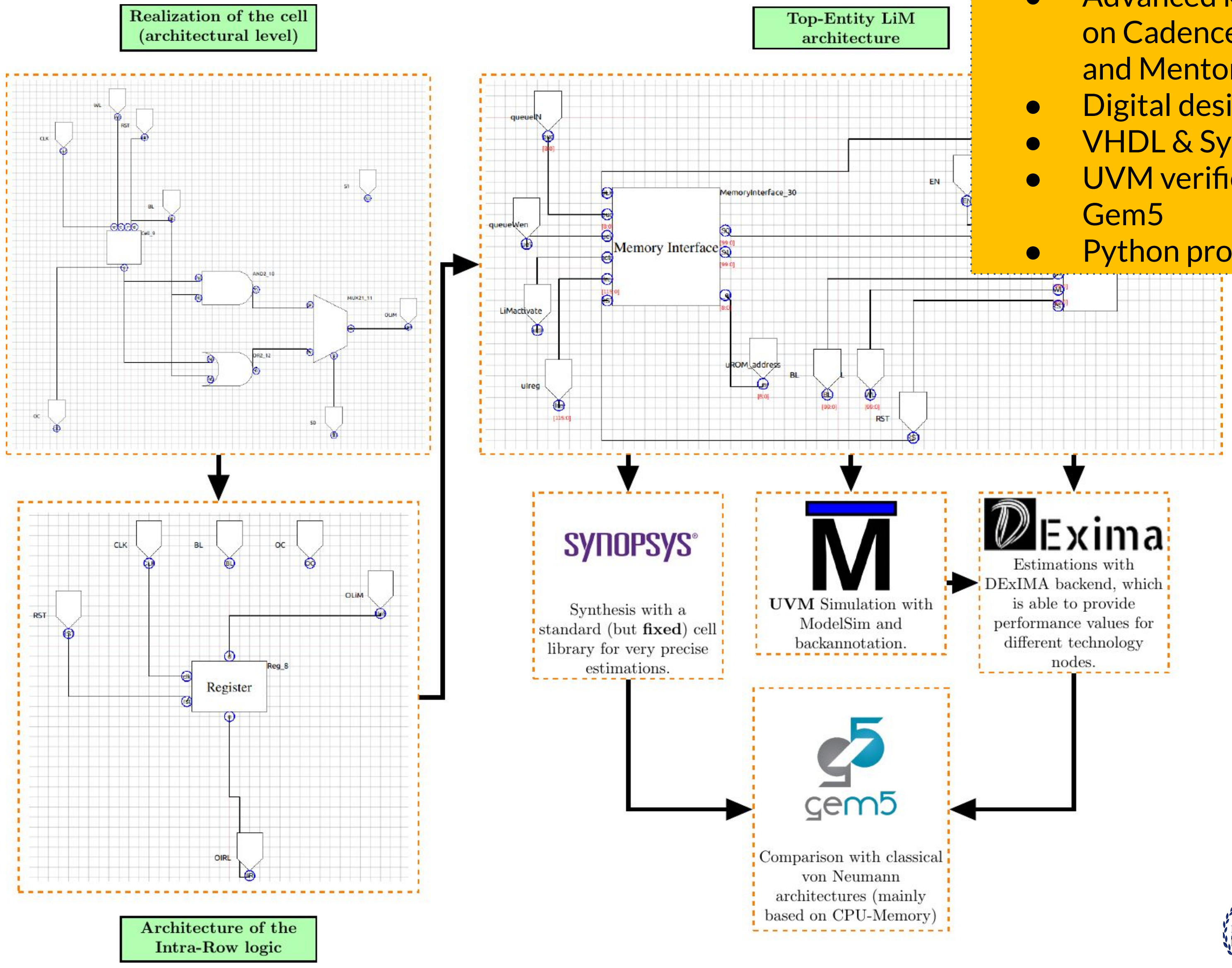


- Several components are part of it:
  - **Manual Design CAD** *(Dexima CAD)*
  - **High-Level Synthesizer**, from C to a LiM architecture *(Octantis)*
  - **LiM Simulator** *(Dexima Simulator)*
  - Technology dependent **Libraries of LiM cells**

T.LiM.1 : Architectural Explorations with Dexima CAD

SKILLS:

- Advanced knowledge on Cadence, Synopsys and Mentor suites
- Digital design flow
- VHDL & SystemVerilog
- UVM verification, Gem5
- Python programming



Synthesis with a standard (but **fixed**) cell library for very precise estimations.

UVM Simulation with ModelSim and backannotation.

Estimations with DExIMA backend, which is able to provide performance values for different technology nodes.

Comparison with classical von Neumann architectures (mainly based on CPU-Memory)

T.LiM.1 : Architectural Explorations with Dexima CAD

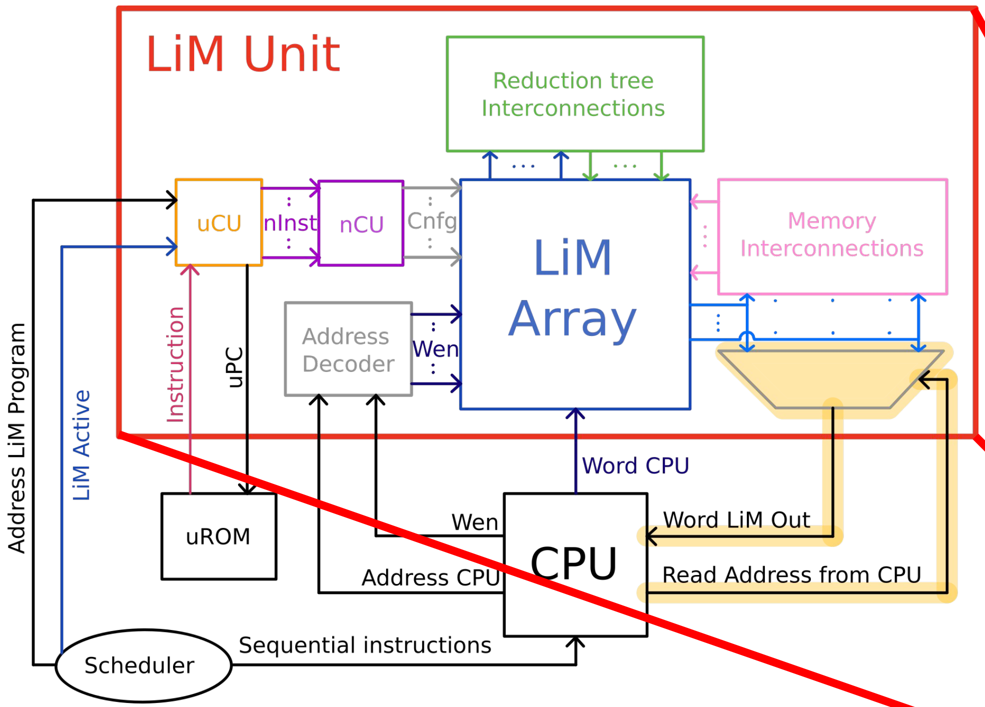
What advantages can be obtained with a LiM architecture with respect to a classical CPU-Memory one?

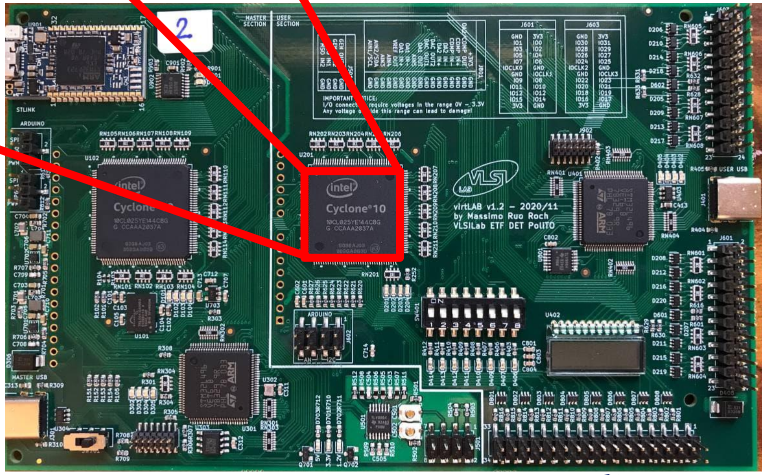
- **Realize** different LiM structures at the architectural level with Dexima CAD
- **Test** their functionality with an embedded UVM testbench
- **Improve** Dexima CAD with new functionalities
- **Estimate** the performance of a LiM architecture with both DExIMA backend and Synopsys/Cadence
- **Compare** the results with a classical von Neumann architecture with Gem5

Politecnico di Torino

T.LiM.2 : FPGA LiM Implementation

SKILLS:
- FPGA and uController programming
- Scripting
- VHDL
- Knowledge on laboratory instruments (Oscilloscope, Multimeter,...)





Evaluate the performance of a LiM architecture on a real board!

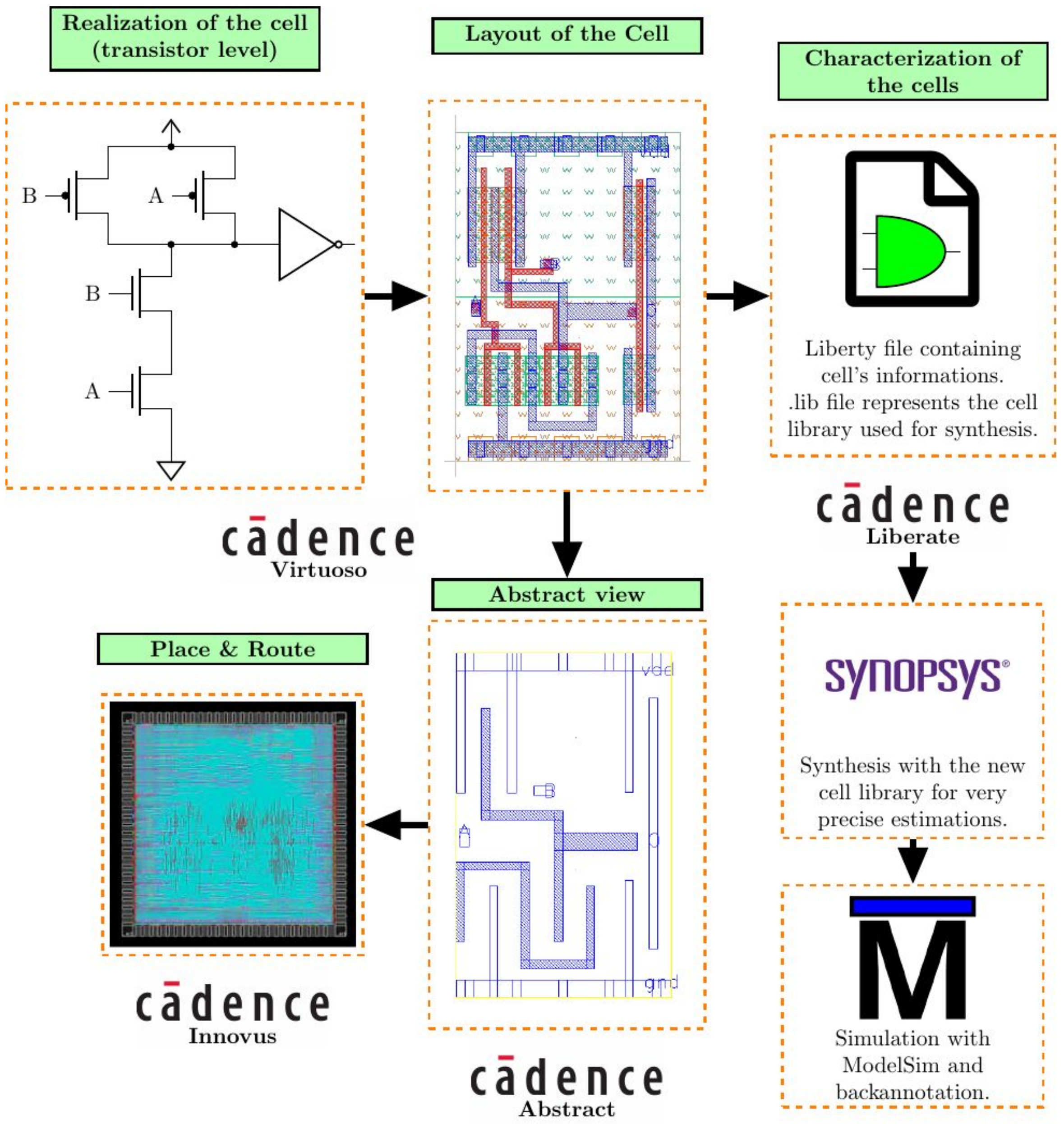
- VirtLab 1.0 from Politecnico di Torino, with 2 Cyclone 10 and 2 STM32L496 microcontrollers

T.LiM.2 : FPGA LiM Implementation

Estimate the impact of the LiM paradigm on a real evaluation board.

- **Realize** different LiM structures at the architectural level. As a starting point, you will use the already existing LiM architectures developed by VLSI Lab.
- **Test** their functionality and **download** the code on the FPGA.
- **Estimate** the performance of a LiM architecture by performing measurements on the circuits.

T.LiM.3 : LiM Toolchain



Liberty file containing cell's informations. .lib file represents the cell library used for synthesis.

Synthesis with the new cell library for very precise estimations.

Simulation with ModelSim and backannotation.

SKILLS:

- Advanced knowledge on Cadence, Synopsys and Mentor suites
- Scripting
- Digital design flow

Precise performance estimations of LiM structures!

Library will be used in standard tools and DExIMA

Politecnico di Torino

T.LiM.3 : LiM Toolchain

Realize a CMOS-Based library for LiM architectures, using the standard Digital Design flow

- **Realize** different LiM cells at the architectural level and **test** their functionality.
- **Design** the LiM cell at transistor-level and **simulate** it with Cadence Virtuoso
- **Layout** of the LiM cell with parasitic extractions
- **Create** a Liberty file of the LiM Cells for synthesis with Synopsys
- **Performance** estimations with Synopsys/Cadence

T.LiM.4 : Octantis

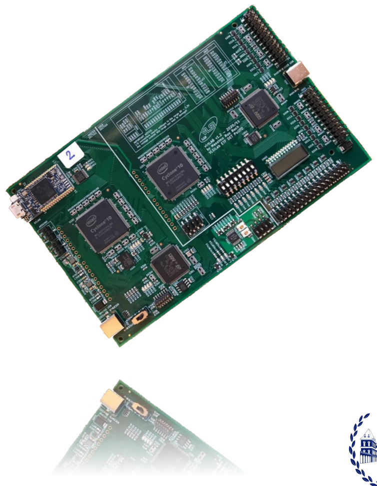
SKILLS:
- C++ programming
- HW design and testing
- FPGA prototyping

Interdisciplinary project (SW and HW)

➔ Starting from an **input C code**, Octantis rearranges it for a direct implementation inside a *Hybrid System provided by a CPU and a LiM Unit*. In particular, an optimal **LiM Unit** is **fully synthesized** by the program.

❖ With the aim of *making the program grow*, add synthesis compatibility to **new LiM architectures**

❖ **Test and prototype** the *accelerator* derived from your work on the VirtLab 1.0 board



T.LiM.5 : Octantis & The LiM Toolchain

SKILLS:

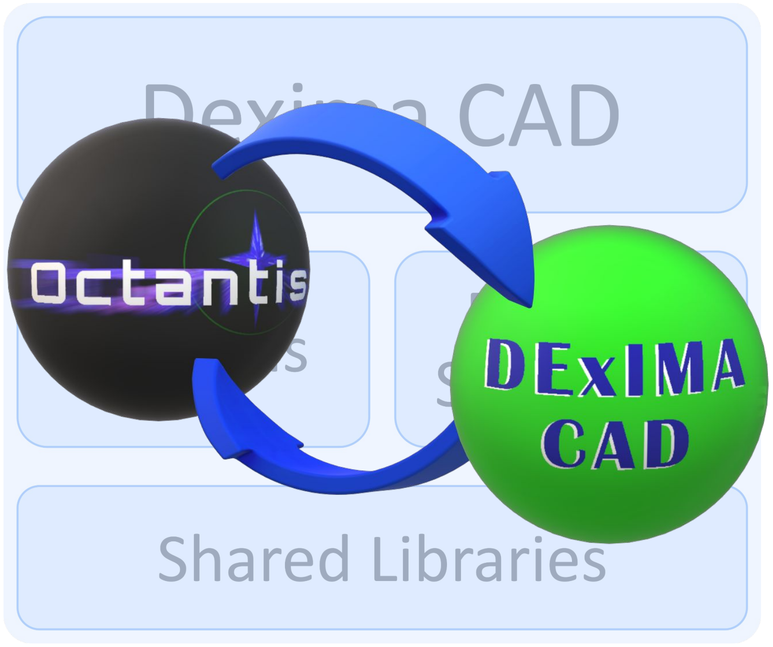
- C++ programming
- Python programming
- HW design and testing
- UVM verification

➔ Starting from an **input C code**, Octantis rearranges it for a direct implementation inside a *Hybrid System provided by a CPU and a LiM Unit*. In particular, an optimal **LiM Unit** is **fully synthesized** by the program.

❖ Expanding the actual **Exploration Capabilities** of the tool and promoting the *integration within the whole LiM Toolchain*

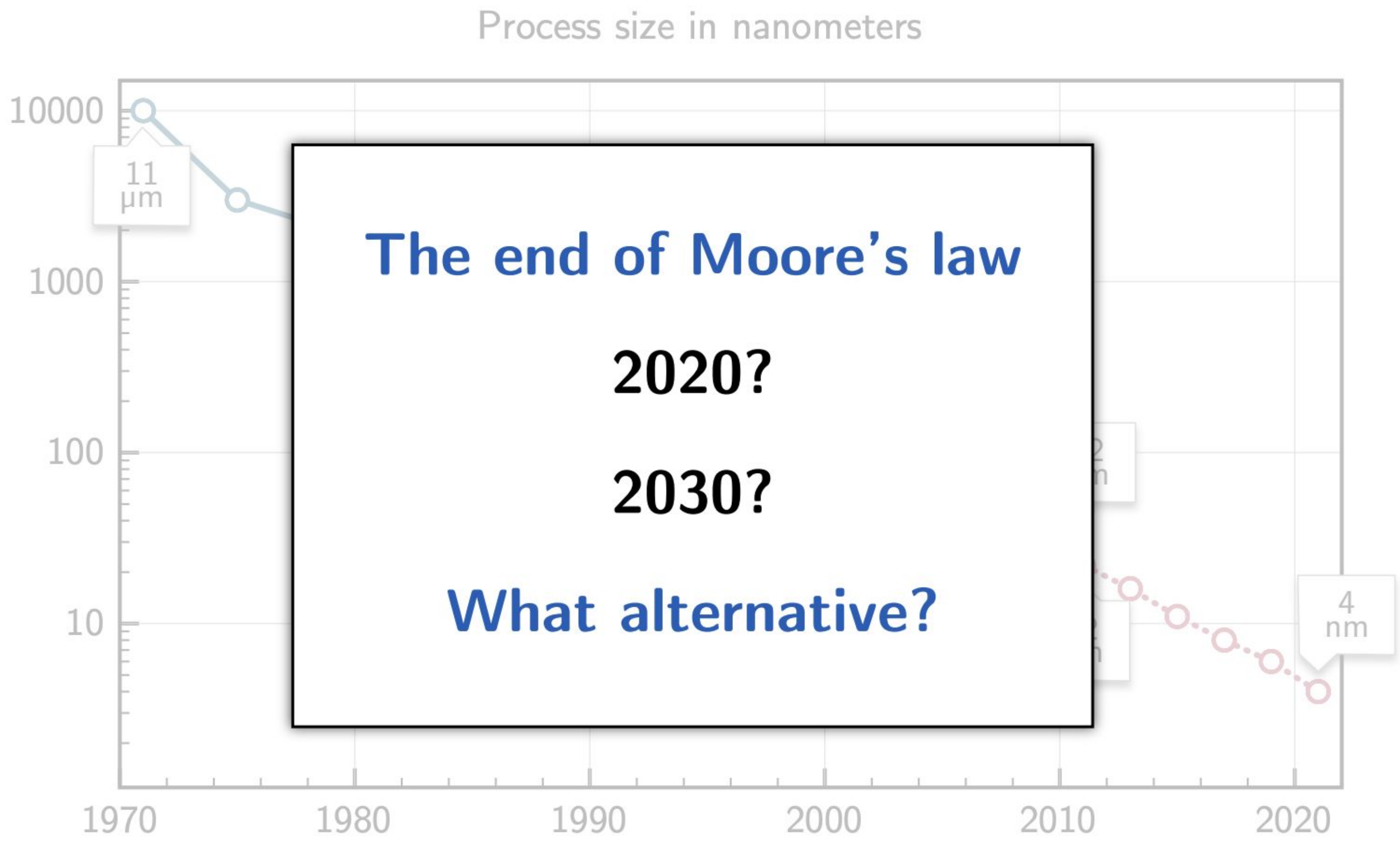
❖ **Add new modules** for improving the collaboration among the tools

❖ The mixed design process will be tested through **advanced verification procedures**



Quantum Computing

Where are we?



The end of Moore's law

2020?

2030?

What alternative?

Analogies with classical computing

Quantum computing and classical computing share the concepts of:

- **Algorithm** as a sequence of transformations
- **State** of a register
  - Bit
  - Qbit
- Quantum computing uses a different computational paradigm from classical one: **operations can be slower/faster** in either model.
- At the end of the computation, some information on the state of the quantum register is obtained by means of a **special measurement** operation.

Some Algorithms

Two most notable algorithms:

- **Shor's** algorithm for integer factorization in polynomial time used for cryptographic purposes [Shor, 1997]
- **Grover's** algorithm for black-box search in O(√n) time [Grover, 1996].
- **Simulation** of complex systems like molecules, economy...

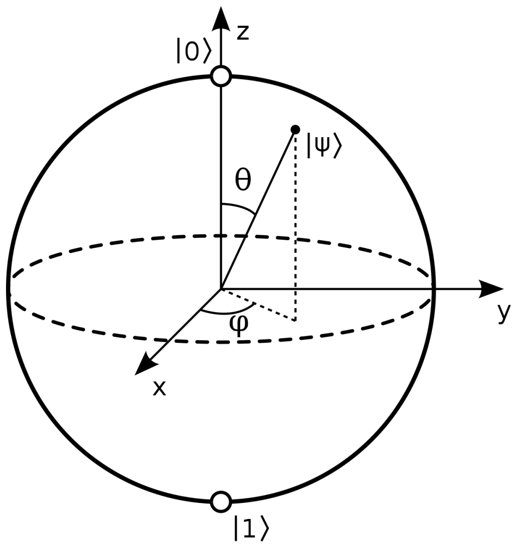
Even if a classical computer can simulate a universal quantum computer, it may not be efficient: some tasks may be exponentially faster in one model of computation versus the other.

Quantum Computing

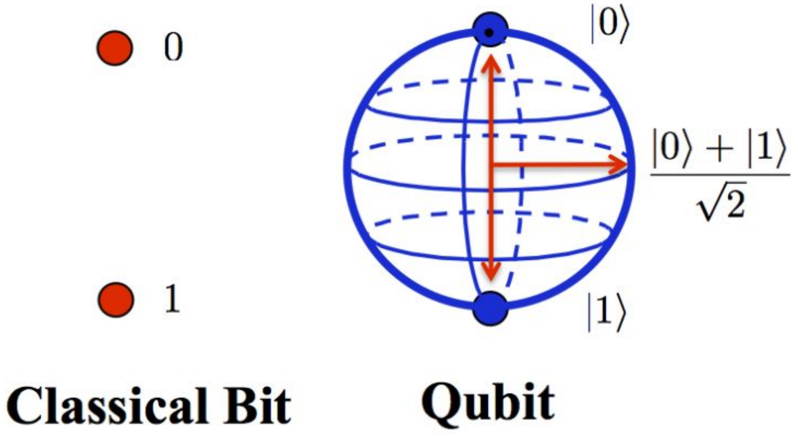
Quantum Computing is a discipline of Information Theory related to the definition of computational routines based on a unit of information, named qubit, encoded onto a quantum physical quantity.

Every computation has three elements:

- **Data** = qubits
- **Operations** = quantum gates (unitary transformations)
- **Results** = measurements



The qubit



- A classical bit can take two different values (0 or 1).
- It is discrete.
- A qubit can “take” infinitely many different values.
- Qubits live in a Hilbert vector space with a basis of two elements that we denote $|0\rangle$ and $|1\rangle$.
- A generic qubit is in a superposition

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

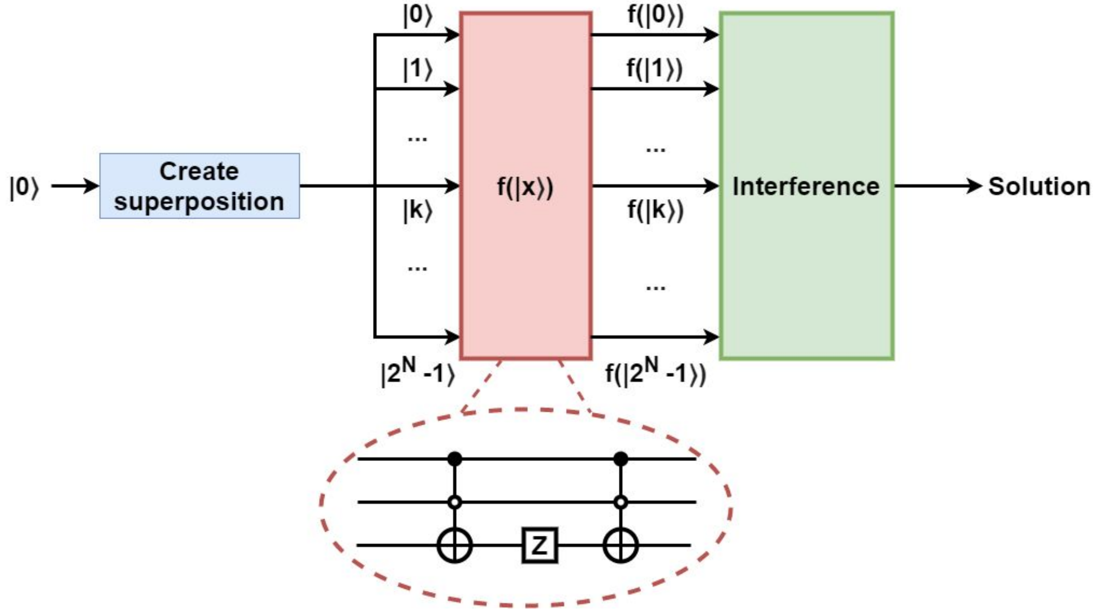
How can we implement a qubit?

Physical system	Qubit state
Photon	Polarization
Electron or nucleus	Spin
Trapped ion	Electronic state
Molecule under NMR	Collective spin
Quantum dot	Charge or spin
Flux qubit	Charge

Quantum Computational Advantage

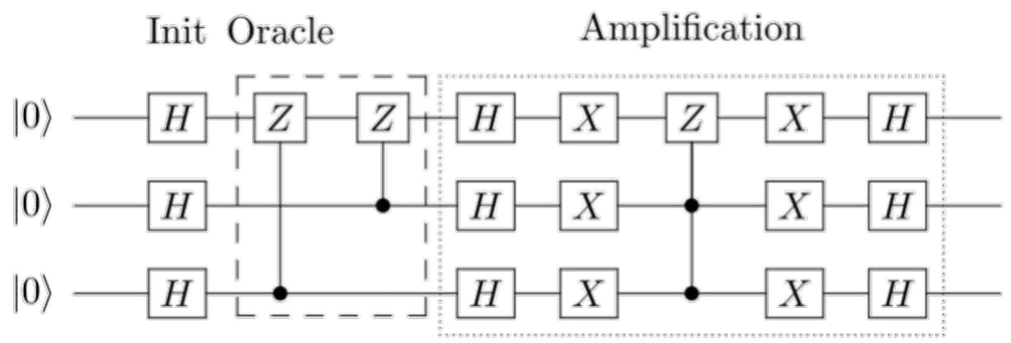
- **Superposition** permits to simultaneously evaluate multiple data
- **Entanglement** can assist in fast converging to the problem's solution.

They both permit to define, for some hard problems, algorithms with computational costs lower than the best corresponding classical ones.

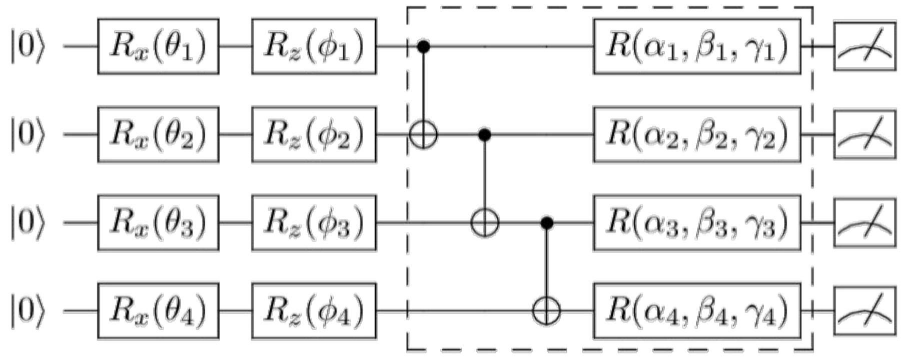


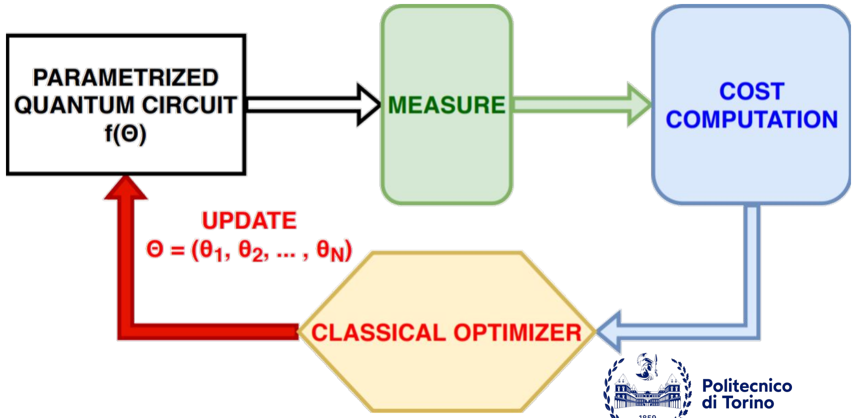
Practical Quantum Circuit strategies

- **Oracular**: circuits thought for labeling a solution and amplifying the probability of measuring it (*e.g.* Grover's search)
- **Variational**: parameterized circuits exploring the whole solutions space, whose quantum gates angles are iteratively updated by a classical optimizer according to a cost function.



https://qiskit.org/textbook/ch-algorithms/grover.html

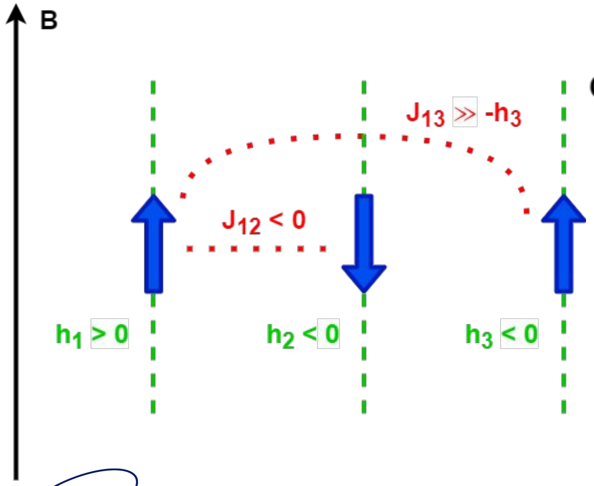

$|0\rangle$
$R_x(\theta_1)$
$R_z(\phi_1)$
$R(\alpha_1, \beta_1, \gamma_1)$
$R_x(\theta_2)$
$R_z(\phi_2)$
$R(\alpha_2, \beta_2, \gamma_2)$
$R_x(\theta_3)$
$R_z(\phi_3)$
$R(\alpha_3, \beta_3, \gamma_3)$
$R_x(\theta_4)$
$R_z(\phi_4)$
$R(\alpha_4, \beta_4, \gamma_4)$

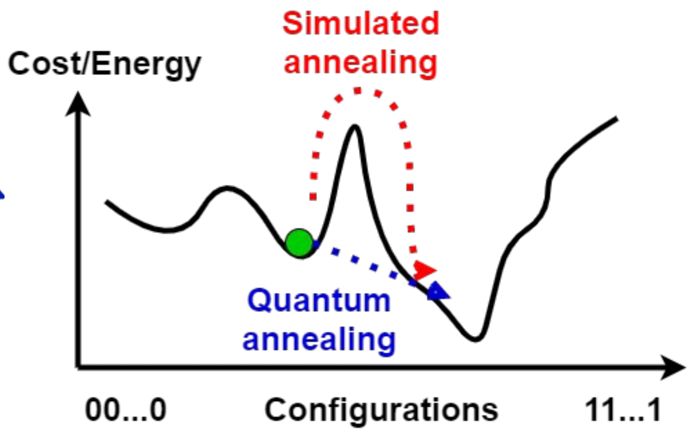

PARAMETRIZED QUANTUM CIRCUIT f(Θ)
MEASURE
COST COMPUTATION
CLASSICAL OPTIMIZER
UPDATE
$\Theta = (\theta_1, \theta_2, \dots, \theta_N)$

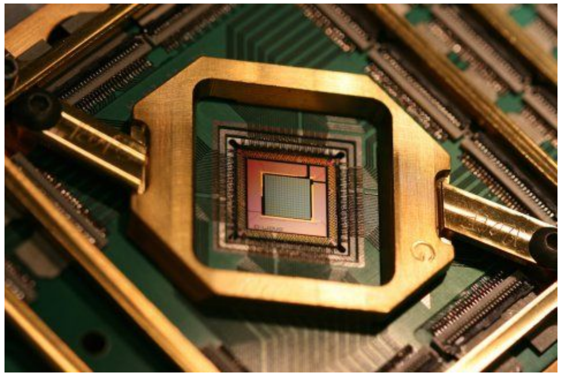
Chen *et al.*, Variational Quantum Circuits for Deep Reinforcement Learning, arXiv:1907.00397

Quantum Annealing

- Specific-purpose Quantum Computing for solving Quadratic Unconstrained Binary Optimization (QUBO) problems, whose cost functions are mapped onto spin Hamiltonians.
- A quantum annealer exploits **tunneling** for achieving the lowest energy spin configuration, corresponding to the optimal solution.
- Quantum annealers are employed for solving **routing or logistic problems** (*e.g.* traffic flow), in **materials engineering** (*e.g.* magnetic properties) and in **Machine Learning**.







https://www.dwavesys.com/media-coverage/techrepublic-d-wave-announces-5000-qubit-fifth-generation-quantum-annealer

Programming languages

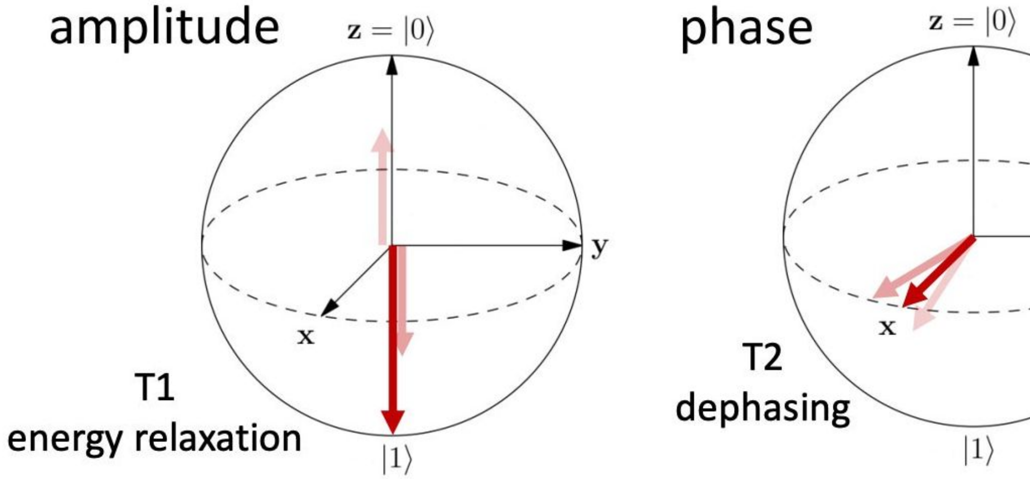
Different frameworks and programming languages:

- qasm
- Qiskit (IBM)
- Cirq (Google)
- Forest/pyqil (Rigetti)
- Q# (Microsoft)
- Ocean (D-Wave)

Limitations of current hardware

- Building a quantum computer is extremely challenging in terms of:
  - Fabrication costs
  - Materials
  - Temperatures (some mK for superconducting qubits, some K for semiconductor qubits)
- Real qubits are always affected by non-ideality phenomena as decoherence, which can affect the reliability of the results of an algorithm on a given quantum computer.

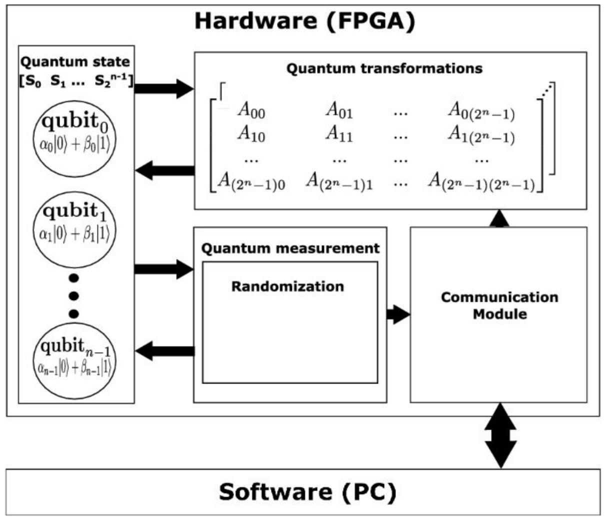
https://www.ibm.com/blogs/research/2017/11/the-future-is-quantum/



https://jonathan-hui.medium.com/qc-how-to-build-a-quantum-computer-with-superconducting-circuit-4c30b1b296cd

Emulators of quantum hardware

- Quantum computers fabricated by some big companies (IBM, Google, D-Wave, *etc.*) are already programmable *via-cloud*.
- Emulation of quantum computers on classical hardware (FPGA, GPU, *etc.*) could be a good approach for having an *on-premises* device behaving like an ideal quantum computer:
  - No decoherence
  - Full connectivity
  - Maintenance costs lower than real quantum computers


Hardware (FPGA)
Quantum state $[S_0\ S_1 \ldots S_{2^{n-1}}]$
qubit_0 $\alpha_0|0\rangle+\beta_0|1\rangle$
qubit_1 $\alpha_1|0\rangle+\beta_1|1\rangle$
qubit_{n-1} $\alpha_{n-1}|0\rangle+\beta_{n-1}|1\rangle$
Quantum transformations
$\begin{bmatrix} A_{00} & A_{01} & \cdots & A_{0(2^n-1)} \\ A_{10} & A_{11} & \cdots & A_{1(2^n-1)} \\ \cdots & \cdots & \cdots & \cdots \\ A_{(2^n-1)0} & A_{(2^n-1)1} & \cdots & A_{(2^n-1)(2^n-1)} \end{bmatrix}$
Quantum measurement
Randomization
Communication Module
Software (PC)

https://www.fujitsu.com/emeia/services/business-services/digital-annealer/what-is-digital-annealer/

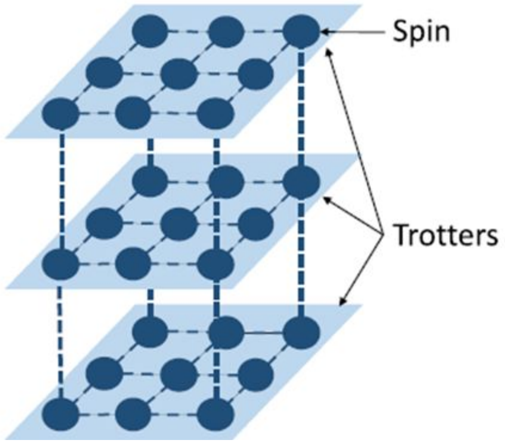
Pilch, J., Długopolski, J. An FPGA-based real quantum computer emulator. *J Comput Electron* **18**, 329–342 (2019). https://doi.org/10.1007/s10825-018-1287-5

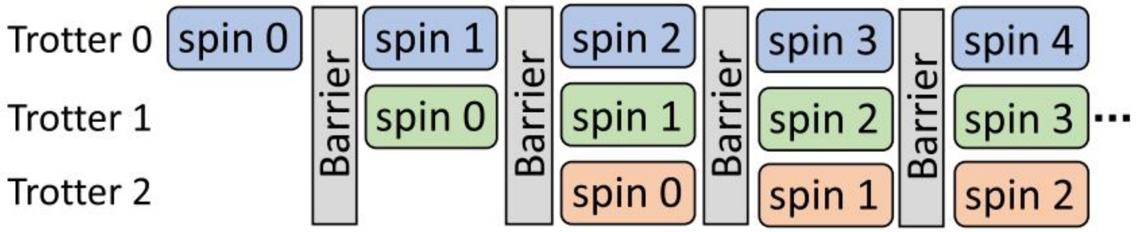
T.QC.1 : Emulation of Quantum Annealing on GPU

SKILLS:
- Quantum annealing
- C/C++ Programming
- GPU programming

Operating steps:

- Analysis of the state of the art algorithms
- Definition of potential improvements
- Software development:
  - GPU emulation
  - Interfaces with frameworks for QUBO/QUSO formulation
- *Scheduling* or *Machine Learning* use cases





Waidyasooriyaet *al.*, [A GPU-Based Quantum Annealing Simulator for Fully-Conncted Ising Models Utilizion Spatial and Temporal Parallelism], IEEE Access, 2020

T.QC.2 : Emulation of Quantum Gate Array On FPGAs

SKILLS:

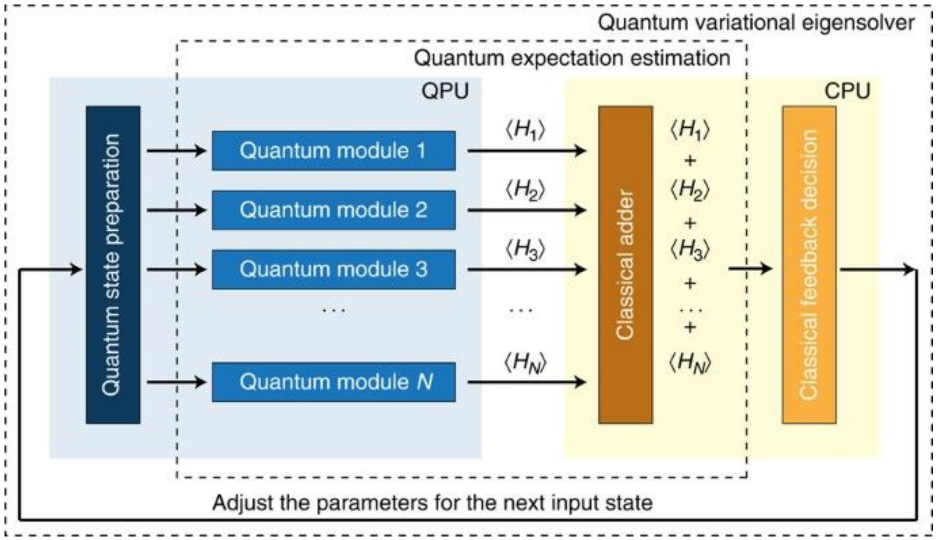
- Quantum computing
- FPGA Design
- uC programming

Feasibility study:

➢ Advantages & Disadvantages of Emulation of **Variational Quantum Algorithms** on *Hybrid Computational Systems*, composed by a uC and a FPGA accelerator.

Operating steps:

- Analysis of the state of the art
- Formulation of an effective VQA model for *Machine Learning*
- Implementation of the algorithm on a FPGA



Quantum variational eigensolver
Quantum expectation estimation
QPU
CPU
Quantum state preparation
Quantum module 1
Quantum module 2
Quantum module 3
...
Quantum module N
$\langle H_1 \rangle$
$\langle H_2 \rangle$
$\langle H_3 \rangle$
$\langle H_N \rangle$
Classical adder
$\langle H_1 \rangle + \langle H_2 \rangle + \langle H_3 \rangle + \dots + \langle H_N \rangle$
Classical feedback decision
Adjust the parameters for the next input state

Peruzzo, A., McClean, J., Shadbolt, P. *et al.* A variational eigenvalue solver on a photonic quantum processor. *Nat Commun* 5, 4213 (2014). https://doi.org/10.1038/ncomms5213

T.QC.3 : Emulation of Quantum Gate Array On GPUs

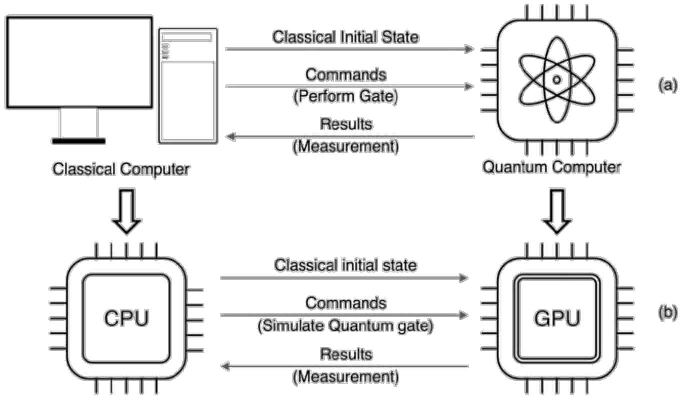
SKILLS:
- Quantum computing
- C/C++ programming
- GPU Programming

Feasibility study:

➢ Advantages & Disadvantages of Emulation of **Variational Quantum Algorithms** on *Hybrid Computational Systems*, composed by a CPU and a GPU.

Operating steps:

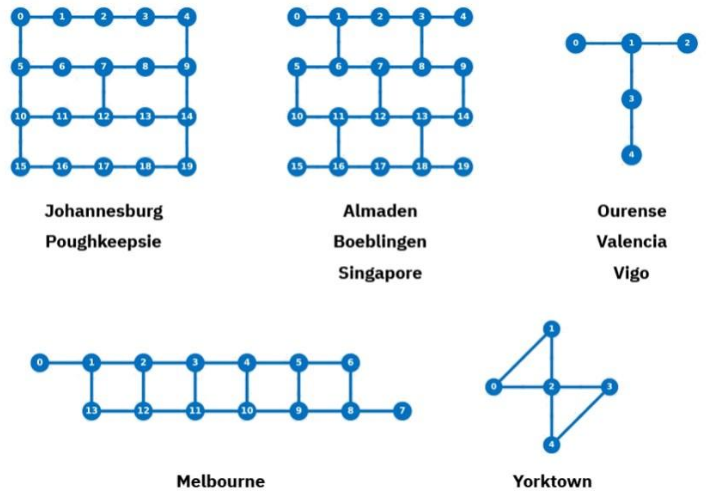
- Analysis of the state of the art
- Formulation of an effective VQA model for *Image Processing*
- Implementation of the algorithm on a GPU



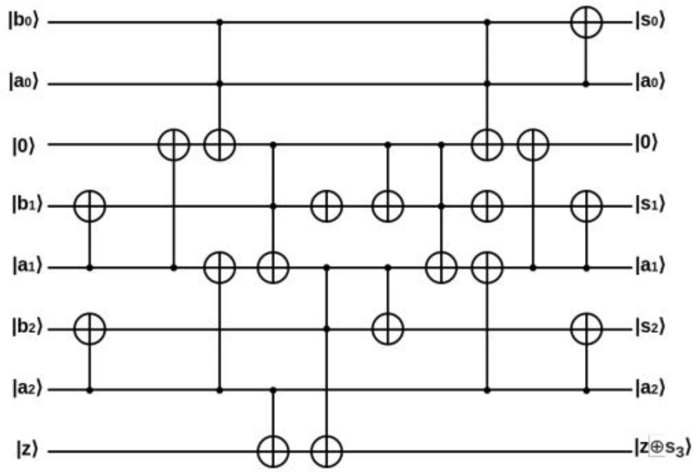
Heng *et al.*, Exploiting GPU-based Parallelism for Quantum Computer Simulation: A Survey, *IEIE Transactions on Smart Processing & Computing*, Vol.9 No.62020.12468 - 476

Quantum Circuit Design

- Real quantum computers have:
  - Specific native gates
  - Potential different decoherence times between qubits
  - Limited qubits connectivity
- Since quantum hardware shows a limited number of non-ideal qubits, the quantum circuit to be executed on a given backend must be critically chosen.
- In order to define a quantum circuit suitable for solving a given problem, simulators/emulators of ideal qubits can be employed without having access to real hardware.



https://www.ibm.com/blogs/research/2019/09/quantum-computation-center/



L.Raggi, *Arithmetic circuits for quantum computing: a software library*, Oct. 2020.

T.QC.4 : Compilation of Quantum Circuits

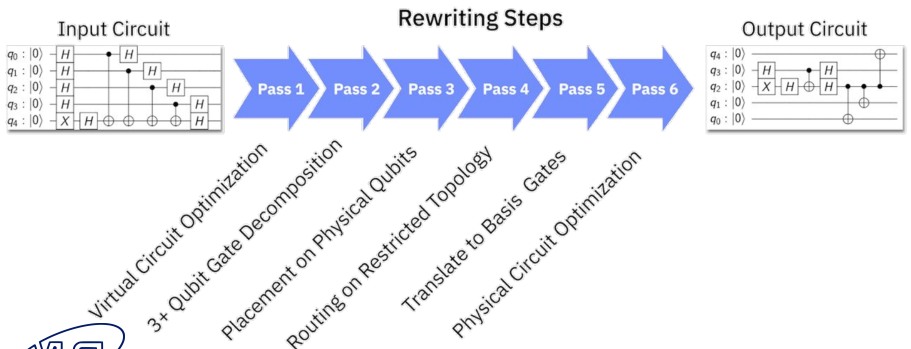
SKILLS:

- Quantum computing
- C/C++ programming
- Python programming

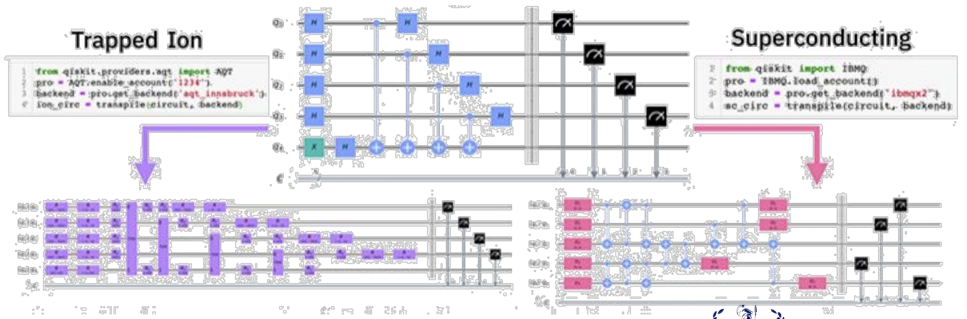
A *quantum compiler* (substantially a **quantum circuit synthesizer**) could enhance the reliability of the circuit to be executed on a real quantum hardware.

Operating steps:

- Study of compilation strategies in the state of the art, with a particular focus on **routing**.
- Analysis of a **compilation toolchain** under development in a thesis ending in July.
- Enhancement of the current compilation toolchain.
- Benchmarking and comparison with other *compilers*


Input Circuit
Rewriting Steps
Output Circuit
Pass 1
Pass 2
Pass 3
Pass 4
Pass 5
Pass 6
Virtual Circuit Optimization
3+ Qubit Gate Decomposition
Placement on Physical Qubits
Routing on Restricted Topology
Translate to Basis Gates
Physical Circuit Optimization

https://qiskit.org/documentation/apidoc/transpiler.html



```
from qiskit.providers.aqt import AQT
pro = AQT.enable_account('1234')
backend = pro.get_backend('aqt_innsbruck')
ion_circ = transpile(circuit, backend)
```

```
from qiskit import IBMQ
pro = IBMQ.load_account()
backend = pro.get_backend('ibmqx2')
sc_circ = transpile(circuit, backend)
```

https://www.ibm.com/blogs/research/2019/11/qiskit-for-multiple-architectures/

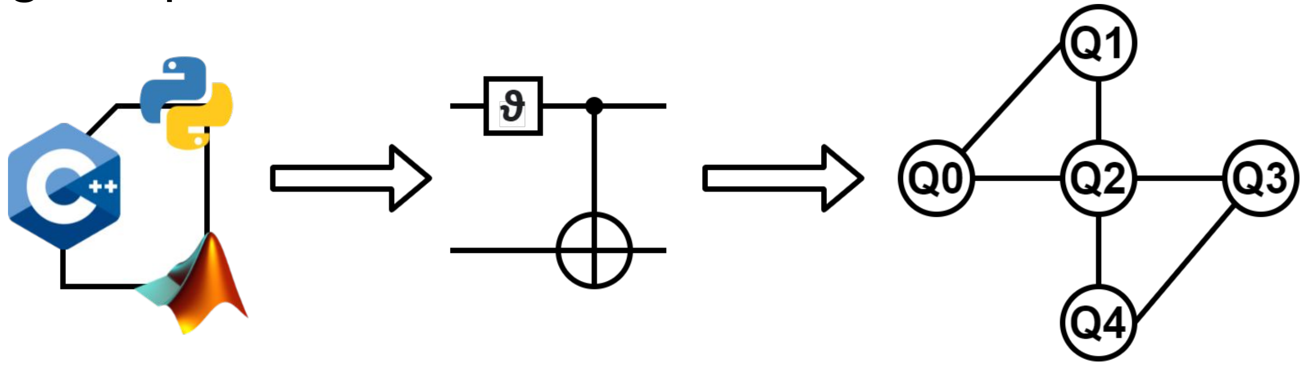
T.QC.5 : Quantum High Level Synthesis

SKILLS:
- Quantum algorithms
- C/C++ programming
- Python programming
- Synthesis Tools

A challenging task of contemporary Quantum Information is defining a general methodology for **automatic construction of quantum circuits from algorithms described classically**.

Operating steps:

- Understanding the main Quantum Computing routines (oracular, variational, *etc.*).
- Analysis of the circuit and compilation libraries under development at VLSI Lab.
- Analysis of High Level Synthesis strategies in Classical Computing.
- Definition of strategies for automatic generation of quantum circuits, according to a problem defined in a high-level language.
- Testing with practical use cases.



CAD Tools for new emerging technology-based paradigms

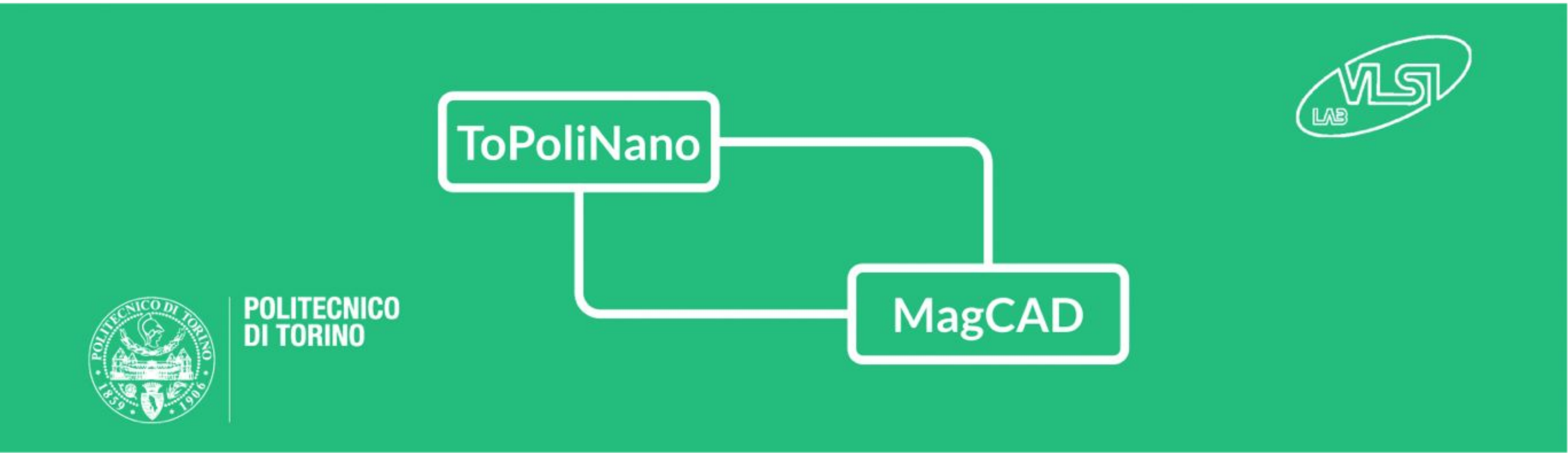
Why?

A change in the technology may imply a deep change in the development of the design tools. Terms of comparison with CMOS technology are needed.

- **Physical simulators** can perform very accurate simulations but they are extremely expensive in terms of computational costs. Indeed, it is not possible to use them to analyze large and complex circuits

- **High level simulators** (like Modelsim) can be used to model the logic behavior of new technology but with this approach the design of large architectures is not manageable.

ToPoliNano Framework

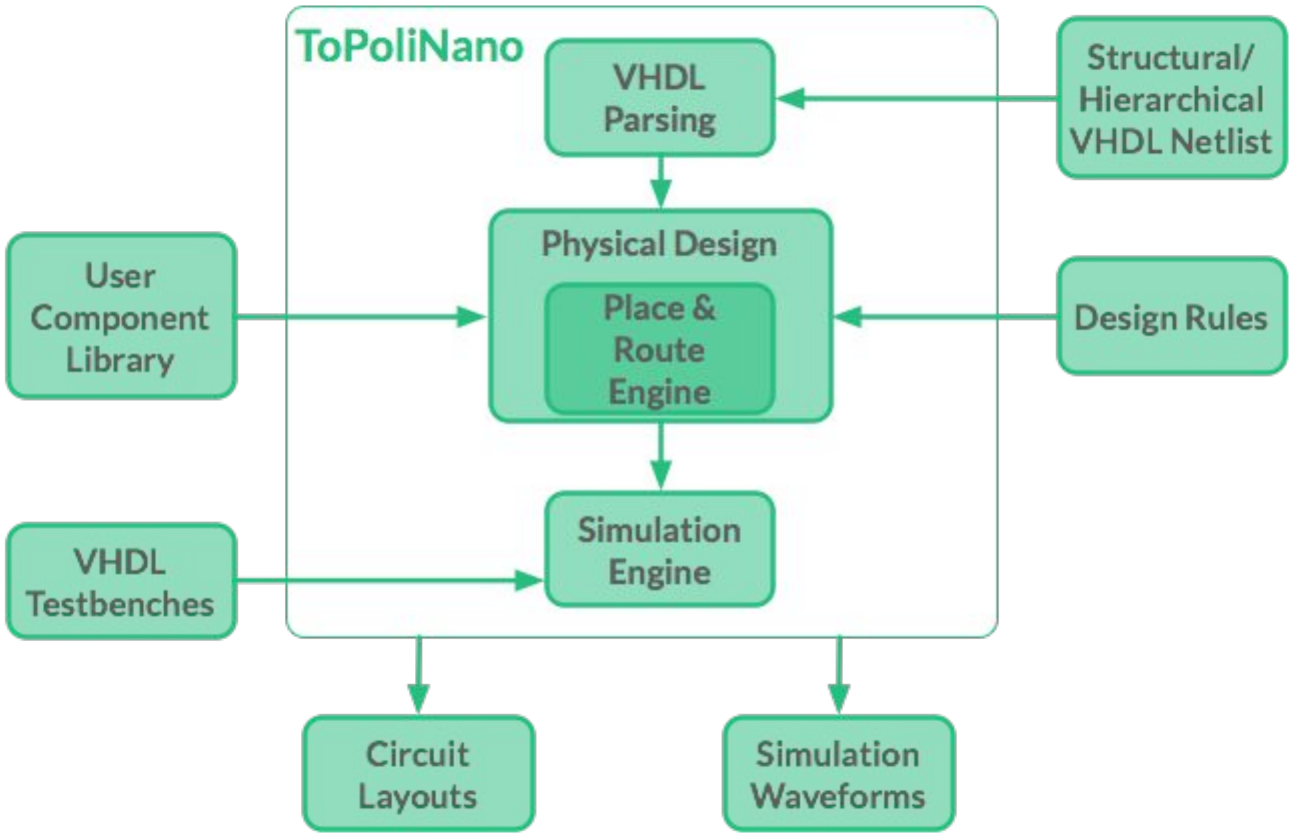


A multi-platform CAD software for emerging nanotechnologies.

Download: https://topolinano.polito.it/

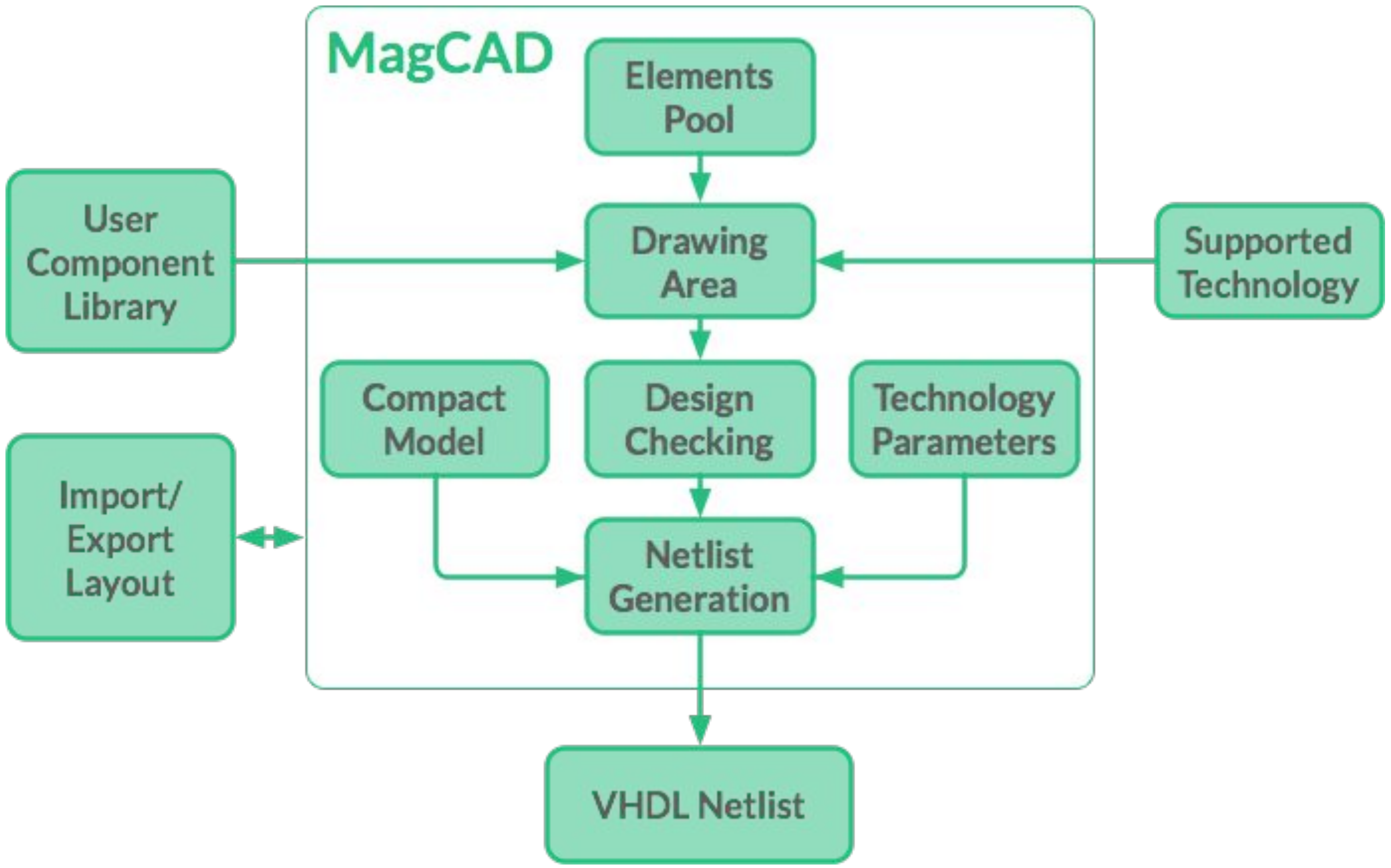
ToPoliNano Framework

ToPoliNano

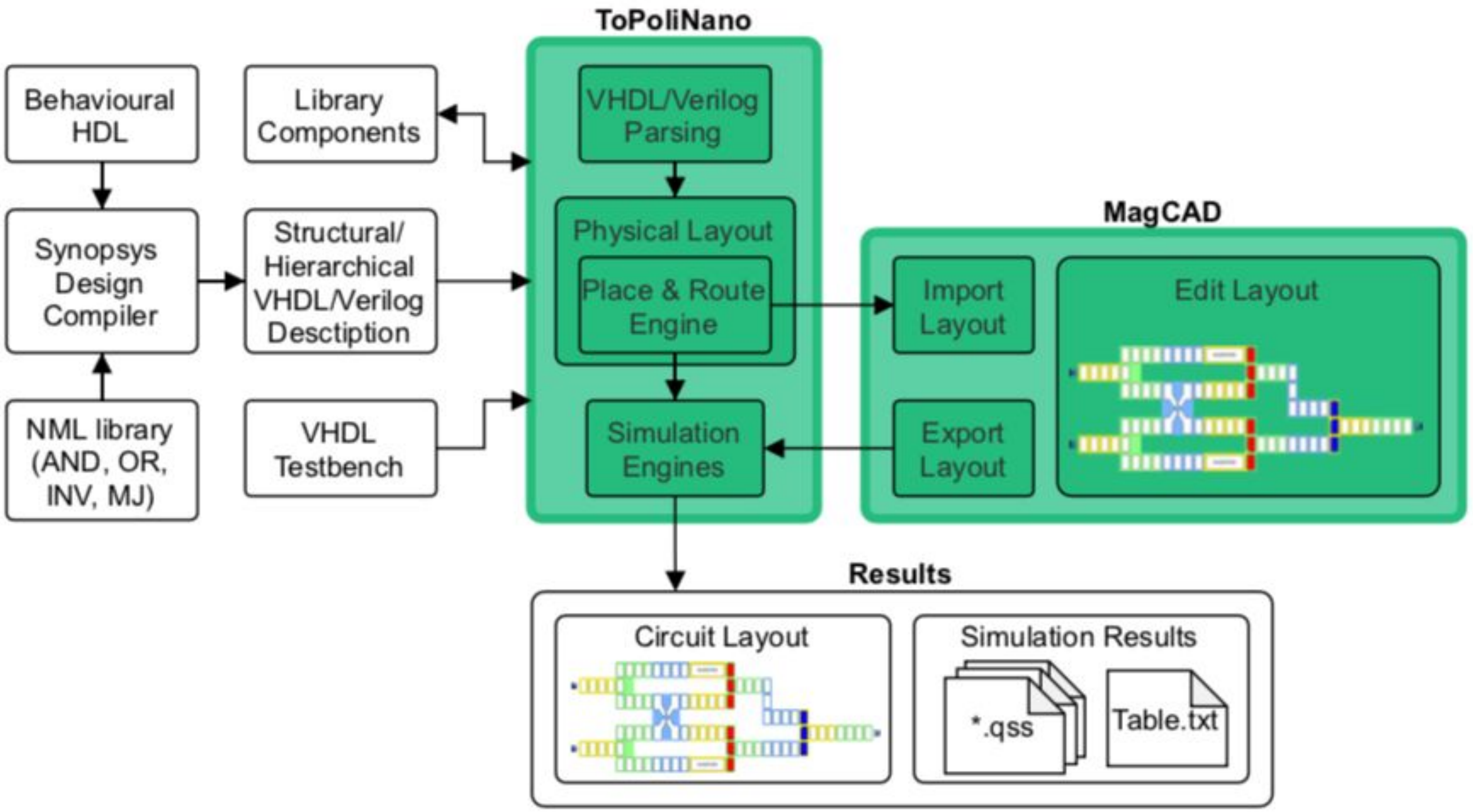


ToPoliNano Framework

MagCAD



ToPoliNano Framework



T.CAD.1 : Hybrid Simulation

➔ Starting from a structural HDL description ToPoliNano can perform physical design and simulation of circuits based on emerging technologies

SKILLS:
- C++ programming
- Existing CAD tool tool usage
- HDL design

❖ Expanding the current **Tool Capabilities** of the tool enabling the possibility to simulate **hybrid CMOS-Emerging technology** circuits

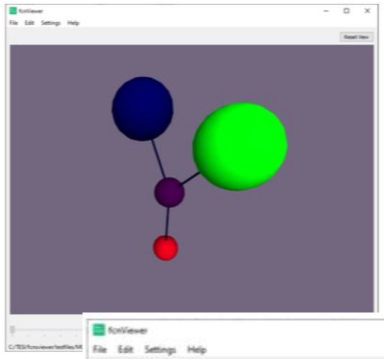
❖ Add domain **translation modules** to interface technologies operating on different domains

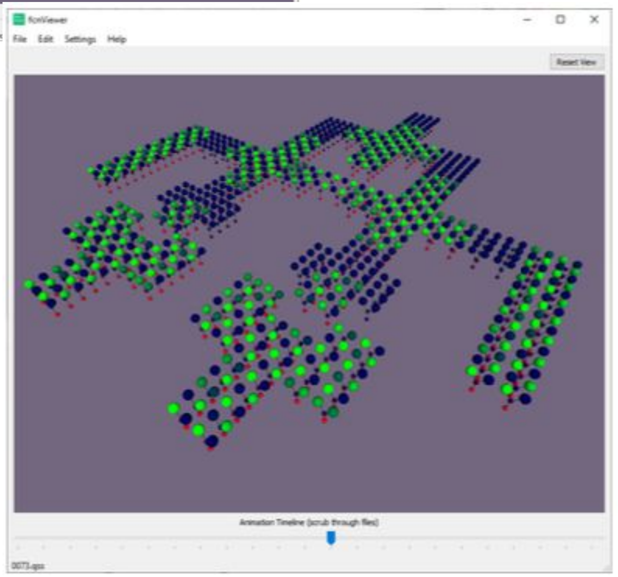
❖ **Automate the design flow**

T.CAD.2 : ToPoliNano Simulator

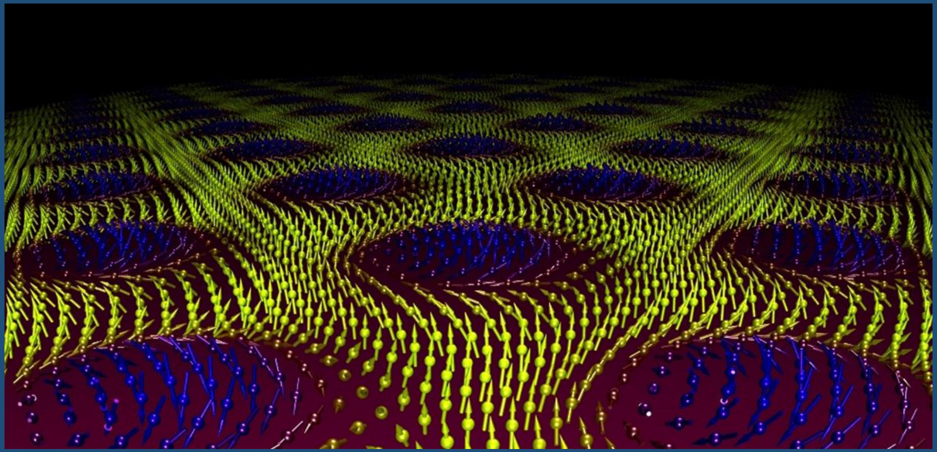
➔ Hybrid simulation requires models of the technologies and ad-hoc simulators

SKILLS:
- C++ programming
- Scripting
- Technology modeling



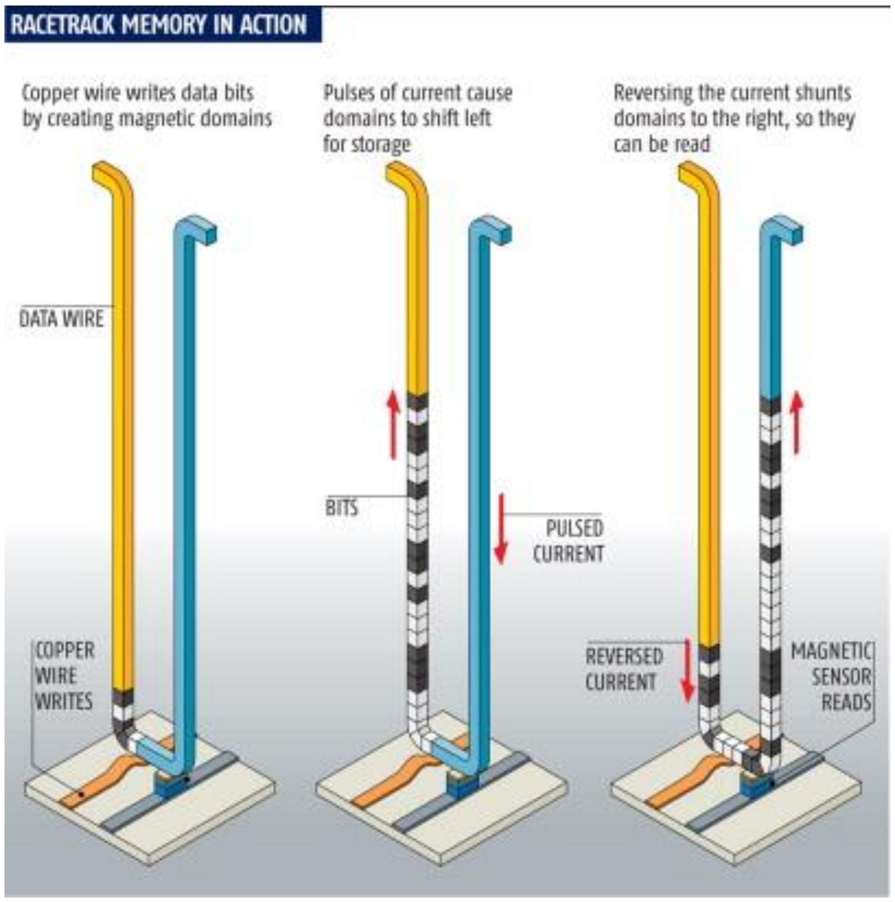


❖ Study models of emerging technologies (e.g. pNML, Skyrmion, spinwaves, etc…)
❖ Extend the integrated simulators within ToPoliNano to enable the simulation of new technologies (e.g. adding the concept of motion)
❖ Validate the simulator with physical simulator data

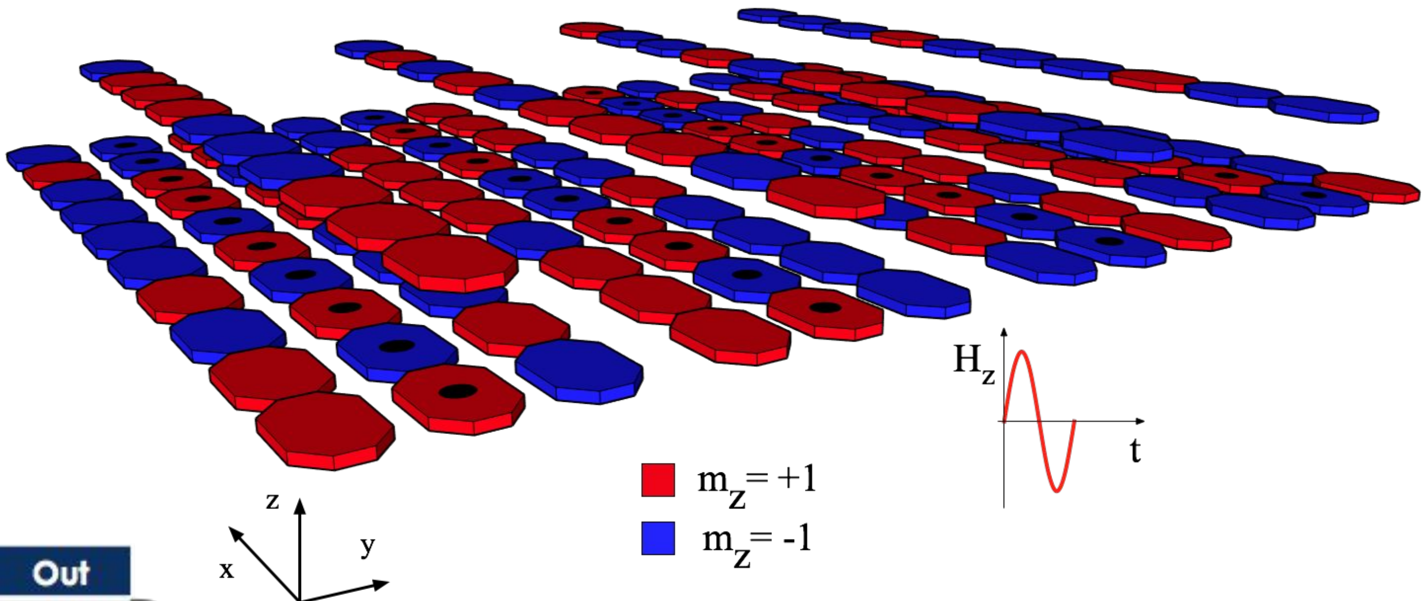


Field Coupled Technologies

Racetrack Memory


RACETRACK MEMORY IN ACTION
Copper wire writes data bits by creating magnetic domains
Pulses of current cause domains to shift left for storage
Reversing the current shunts domains to the right, so they can be read
DATA WIRE
BITS
PULSED CURRENT
COPPER WIRE WRITES
REVERSED CURRENT
MAGNETIC SENSOR READS

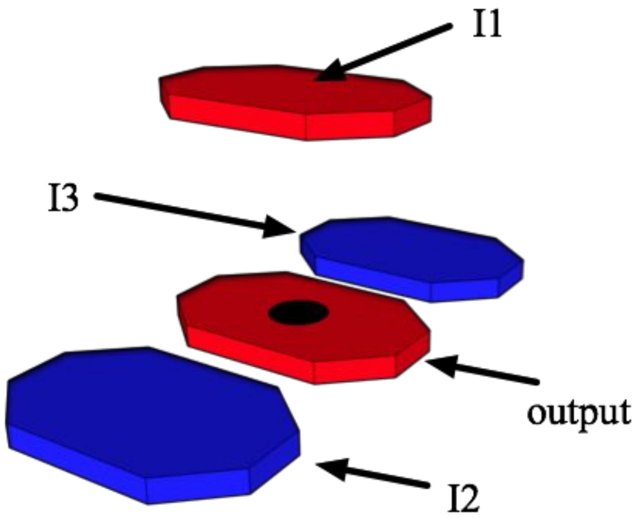
Racetrack Memory



I1	I2	I3	Out
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

NOR

NAND



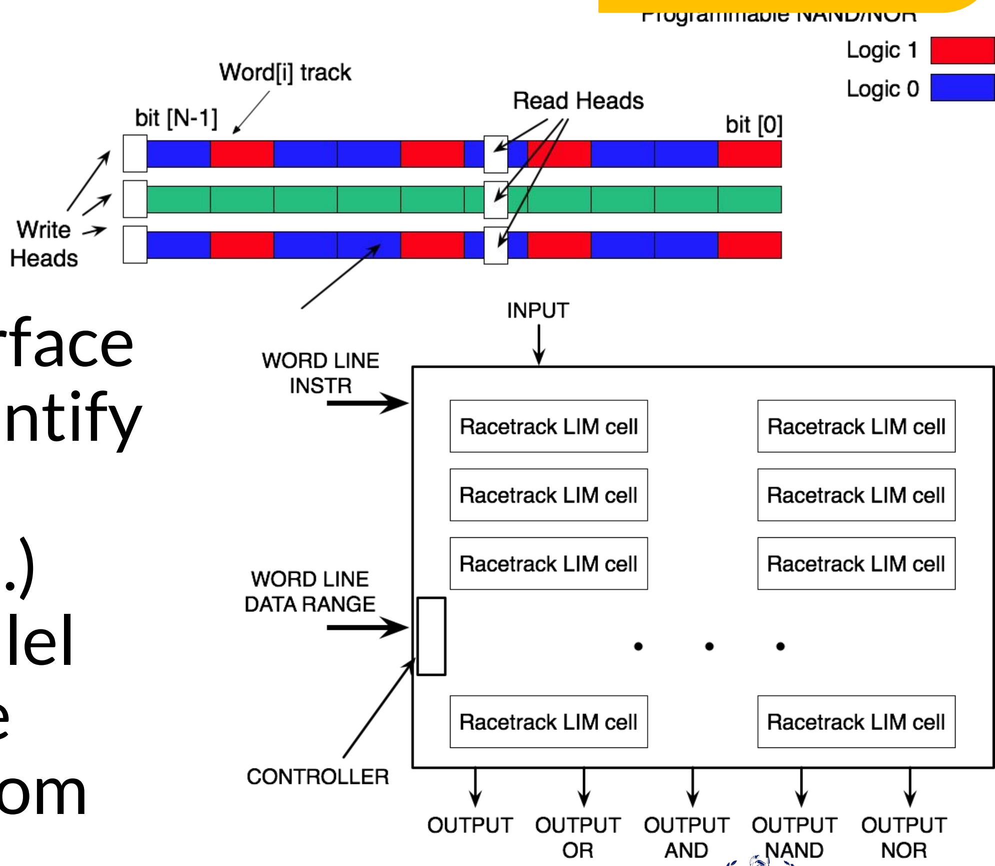
T.MAG.1 : Racetrack Memory Architecture

SKILLS:
- knowledge on Synopsys and Mentor suites
- Scripting
- Digital design flow

➔ Racetrack is an extremely dense

Memory with computing capabilities

❖ Develop a high level memory model with standard memory interface
❖ Perform analysis to identify the best memory organization (size, etc...)
❖ Implement highly parallel algorithm and compare performance with custom memories



Questions?

Stay tuned on
www.vlsilab.polito.it

The theses presented here are ready now and available in the next few months. However other related or prosecutions will be available later.